DGMD E-17: Homework Assignment A3

Matthew Parker - Harvard University - Spring 2022

1. Assignment Details

Assignment 3
Use ROS Environment and Gazebo to simulate a RasPi Robot

setup the ROS environment:

```
matt@matt-VirtualBox:~$ source /opt/ros/noetic/setup.bash
```

Start roscore

```
matt@matt-VirtualBox:~$ roscore
... logging to /home/matt/.ros/log/cb7e4dc2-a7b5-11ec-8c3d-39abf98c29c4/roslaunch-matt-VirtualBox-31057.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://matt-VirtualBox:35985/
ros_comm version 1.15.14


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [31067]
ROS_MASTER_URI=http://matt-VirtualBox:11311/

setting /run_id to cb7e4dc2-a7b5-11ec-8c3d-39abf98c29c4
process[rosout-1]: started with pid [31077]
started core service [/rosout]
```

Then create new package.

```
matt@matt-VirtualBox:~/catkin_ws/src$ catkin_create_pkg ros_gazebo_rviz1 rospy std_msgs
Created file ros_gazebo_rviz1/package.xml
Created file ros_gazebo_rviz1/CMakeLists.txt
Created folder ros_gazebo_rviz1/src
Successfully created files in /home/matt/catkin_ws/src/ros_gazebo_rviz1. Please adjust the values in package.xml.
```

Now build package

```
matt@matt-VirtualBox:~/catkin_ws$ catkin_make
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~  traversing 2 packages in topological order:
~~  - ros_101_gazebo_rviz
~~  - ros_gazebo_rviz1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+++ processing catkin package: 'ros_101_gazebo_rviz'
==> add_subdirectory(ros_gazebo_rviz)
+++ processing catkin package: 'ros_gazebo_rviz1'
==> add_subdirectory(ros_gazebo_rviz1)
Configuring done
Generating done
Build files have been written to: /home/matt/catkin_ws/build
```

Don’t forget to add development setup to ROS path:

```
matt@matt-VirtualBox:~/catkin_ws$ . ~/catkin_ws/devel/setup.bash
```

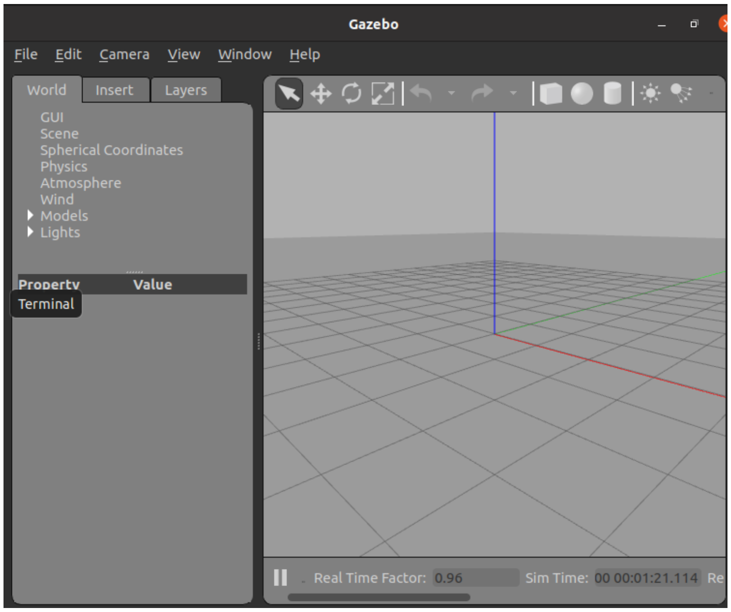
3. Start Gazebo Simulator

Run command to confirm gazebo is installed and running correctly.

```
matt@matt-VirtualBox:~$ rosrun gazebo_ros gazebo
[ INFO] [1647716061.504892412]: Finished loading Gazebo ROS API Plugin.
[ INFO] [1647716061.511747718]: waitForService: Service [/gazebo/set_physics_proper
ties] has not been advertised, waiting...
[ INFO] [1647716062.785616129]: waitForService: Service [/gazebo/set_physics_proper
ties] is now available.
[ INFO] [1647716062.865494236]: Physics dynamic reconfigure ready.
```

gazebo11.desktop
“Gazebo” is ready

Gazebo GUI View

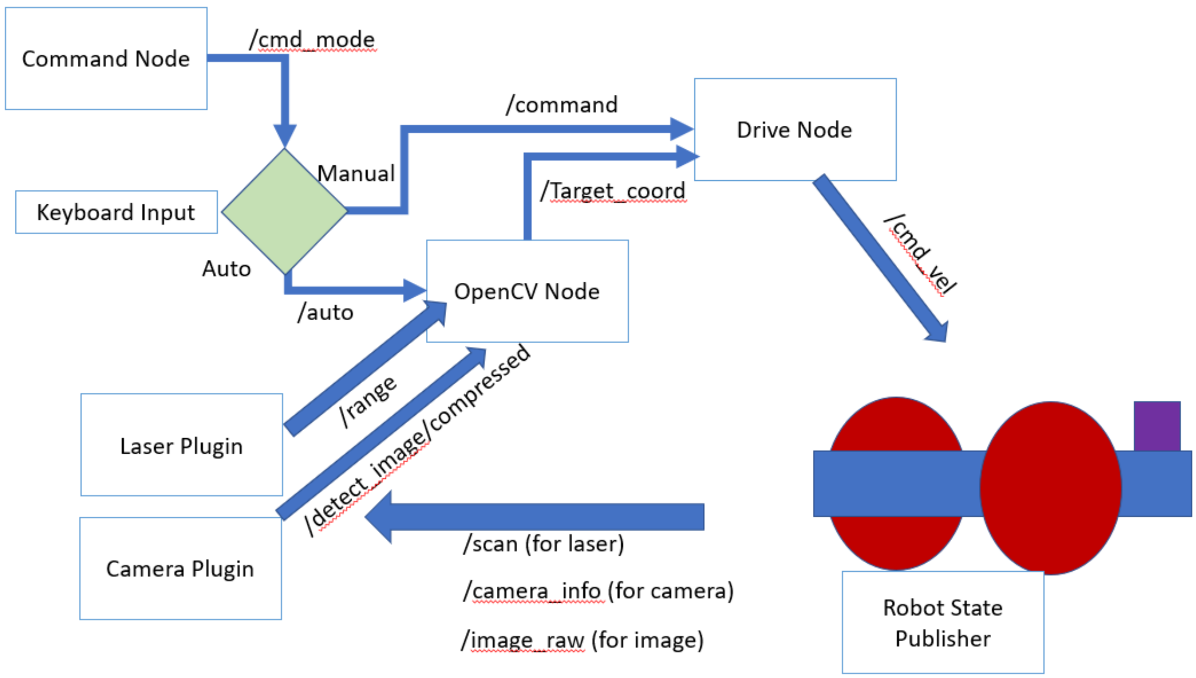


Verify all topics are publishing correctly

```
matt@matt-VirtualBox:~/catkin_ws$ rostopic list
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/rosout
/rosout_agg
```

4. Configure Robot Files

First, let's look at the overall data flow diagram for this project:



A description of each file is provided below, with the full source code copied to the appendix:

1. Command Node
    a. Command_node.py file
    b. Similar to what was used in the first HW, but with the addition of the "a" key sending an "auto" command to enable autonomous operations.
2. OpenCV Node
    a. Image_processor.py file
    b. This is the main brains behind the driving. The file takes in an image from the Gazebo Camera Plugin instead of the webcam from the RPi example. From here it performs the same target detection and identification of target coordinates. This node then publishes the coordinates to the drive node for turning decisions.
3. Drive Node
    a. Drive_node.py file
    b. Again, this is similar to the provided code, but we had to modify the inputs and outputs to publish to ROS topics instead of publishing to RPi GPIO pins directly since there is no hardware in this simulation. Additionally, we had to add direct linear and angular velocity commands for the /cmd_vel topic to send the correctly formatted commands to the Gazebo differential drive plugin module.
4. Robot State Publisher
    a. Node is started when the xacro_robot_gazebo.launch file is executed. This handles the incoming velocity commands and then publishes the robot state, including information from sensors.
5. Camera Plugin

   a. This node is also started when the xacro_robot_gazebo.launch file is executed. This node is taking in raw camera images and metadata from camera_info to provide a pre-processed compressed image to be sent to the computer vision module.
   b. Note: the Laser Plugin was shown because we did optionally add a laser tracker to this module. It's not directly required for the current form of autonomous driving, but it will be useful when the robot needs to perform more complex movements like collision avoidance.

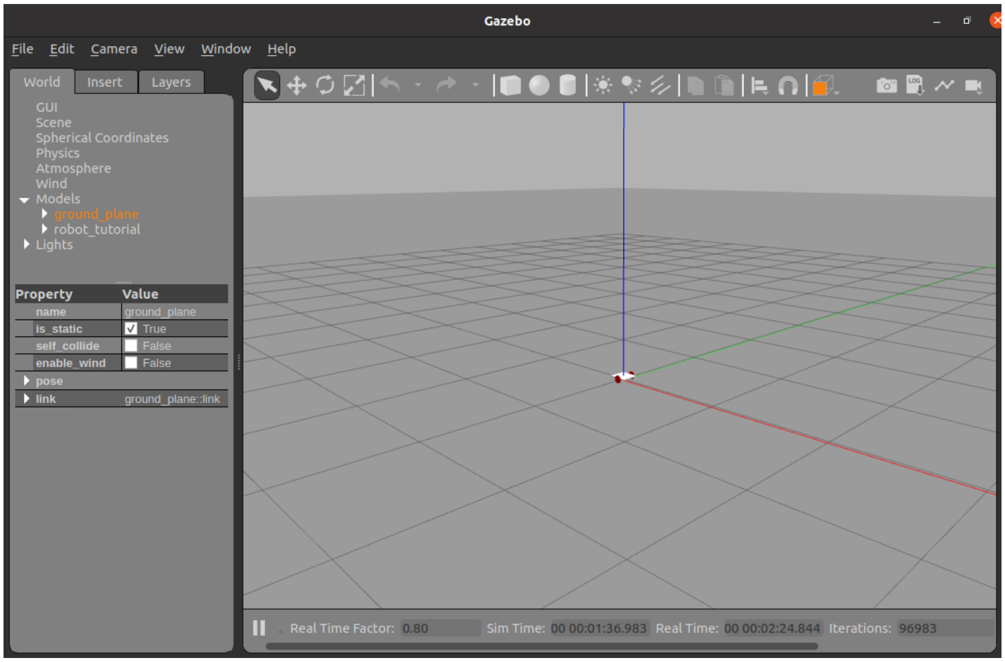
The package file structure looks like this:

- Ros_gazebo_rviz1
  - CMakeLists.txt
  - Package.xml
  - Launch
    - Robot_gazebo.launch
  - Src
    - Sonar.py (not required, used for laser scanner)
  - Urdf
    - Robot_tutorial.xacro
    - Robot_tutorial_gazebo.xacro
    - Camera_gazebo.xacro
    - laser_gazebo.xacro
  - Scripts
    - Command_node.py
    - Drive_node.py
    - Image_processor.py
  - Rviz
    - main.rviz

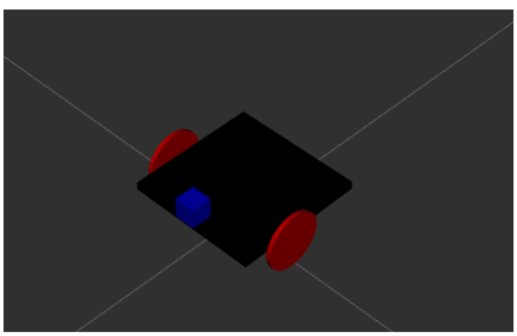
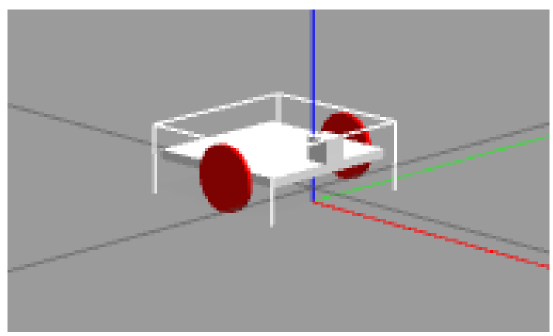
5. Load Robot Into Simulator

Let's try the original robot, verify we can load into Gazebo:

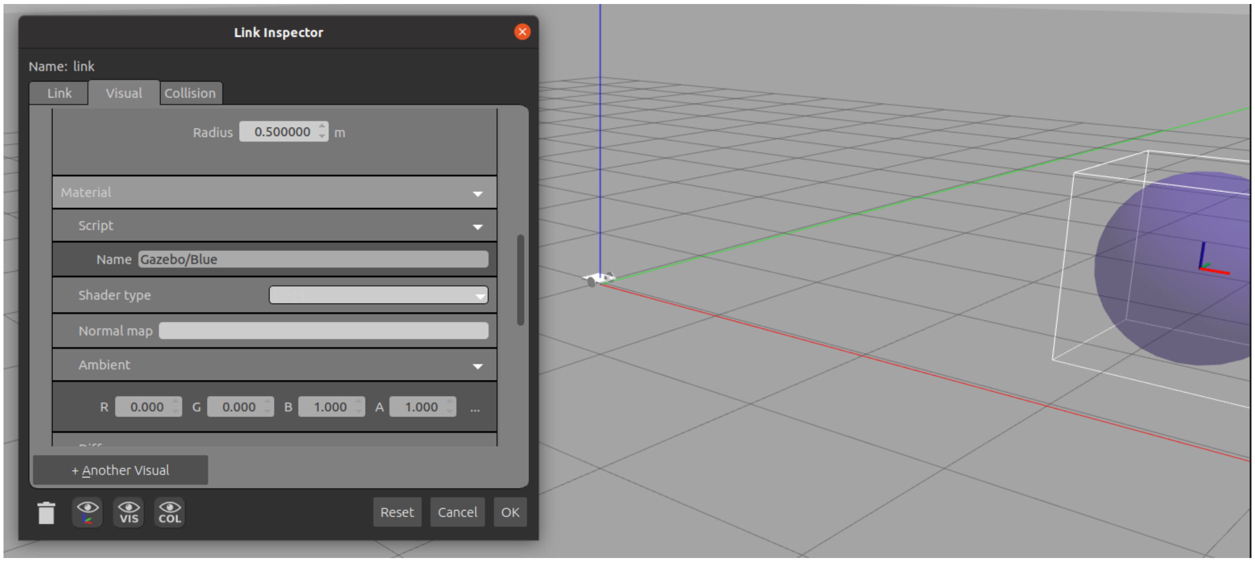
```
matt@matt-VirtualBox:~/catkin_ws/src/ros_gazebo_rviz1/launch$ roslaunch robot_ga
zebo.launch
... logging to /home/matt/.ros/log/7ec4b41c-ad3c-11ec-a0c6-d5e8643e799d/roslaunc
h-matt-VirtualBox-3219.log
Checking log directory for disk usage. This may take a while.
```



Now after adding the camera to the front of the robot:

Rviz	Gazebo
	

Add a blue sphere to the world:



6. Prepare to Run

For this task we will utilize the command_node and drive_node files used for the original robot car controller.
These two files were added to the ros_gazebo_rviz1 package in the scripts folder. Now make them executable:

```
matt@matt-VirtualBox:~/catkin_ws/src/ros_gazebo_rviz1/scripts$ chmod +x command_node.py
matt@matt-VirtualBox:~/catkin_ws/src/ros_gazebo_rviz1/scripts$ chmod +x drive_node.py
```

Add new “image_processor.py” script, based on the OpenCV code but expanded to take in images from Gazebo instead of a webcam. Also make executable.

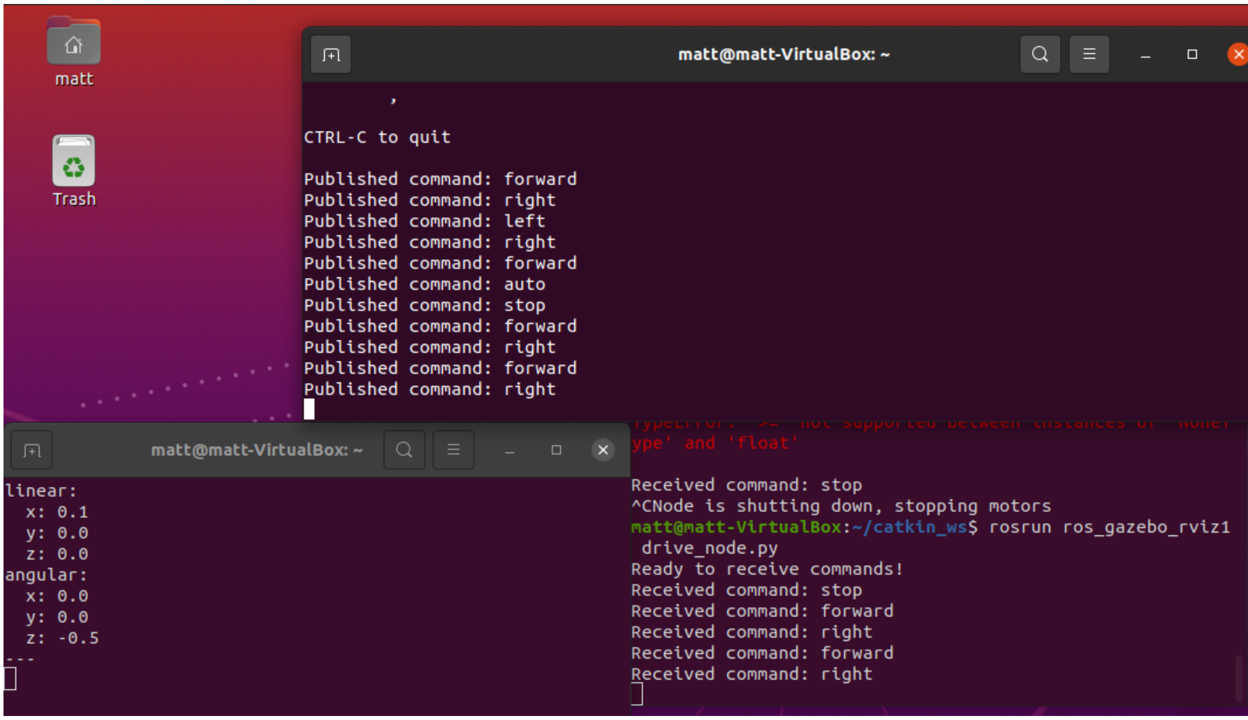
```
matt@matt-VirtualBox:~/catkin_ws/src/ros_gazebo_rviz1/scripts$ chmod +x image_processor.py
```

Verify command_node is working:

```
matt@matt-VirtualBox:~$ rosrun ros_gazebo_rviz1 command_node.py

Reading from the keyboard and publishing to /command!
---------------------------
Moving around:
        i
   j    k    l
        ,

CTRL-C to quit

Published command: forward
Published command: stop
Published command: right
Published command: left
Published command: backward
Published command: auto
```

Check to see everything is publishing correctly:

- Camera info topic
  - ```
    matt@matt-VirtualBox:~$ rostopic echo -c /camera_info
    ```
  - ```
    header:
      seq: 194
      stamp:
        secs: 29829
        nsecs: 264000000
      frame_id: "camera_link"
    height: 240
    width: 240
    distortion_model: "plumb_bob"
    D: [0.0, 0.0, 0.0, 0.0, 0.0]
    K: [143.01092508042584, 0.0, 120.5, 0.0, 143.0109250804258
    4, 120.5, 0.0, 0.0, 1.0]
    R: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
    P: [143.01092508042584, 0.0, 120.5, -10.01076475562981, 0.
    0, 143.01092508042584, 120.5, 0.0, 0.0, 0.0, 1.0, 0.0]
    binning_x: 0
    binning_y: 0
    roi:
      x_offset: 0
      y_offset: 0
      height: 0
      width: 0
      do_rectify: False
    ```
- Do some manual robot motions to confirm the /cmd_vel
  - Great, can see the linear and angular velocities being sent correctly to the robot

- 

Now start the image processor node:

```
matt@matt-VirtualBox:~$ rosrun ros_gazebo_rviz1 scripts/image_processor.py
Start Listening to receive Gazebo Data
```

Let's see that everything is available:

1. Check nodes:
    a.
    ```
    matt@matt-VirtualBox:~$ rosnode list
    /detection_node_3041_1648823686485
    /gazebo
    /gazebo_gui
    /joint_state_publisher
    /keyb_commander_2710_1648822915868
    /motor_driver_2783_1648823011328
    /robot_state_publisher
    /rosout
    /rviz
    matt@matt-VirtualBox:~$
    ```
    b. Good we have everything expected
        i. Detection node is for image_processor
        ii. Gazebo and Rviz are both running
        iii. Keyb_commander and motor_driver are running for command_node and drive_nodes respectively

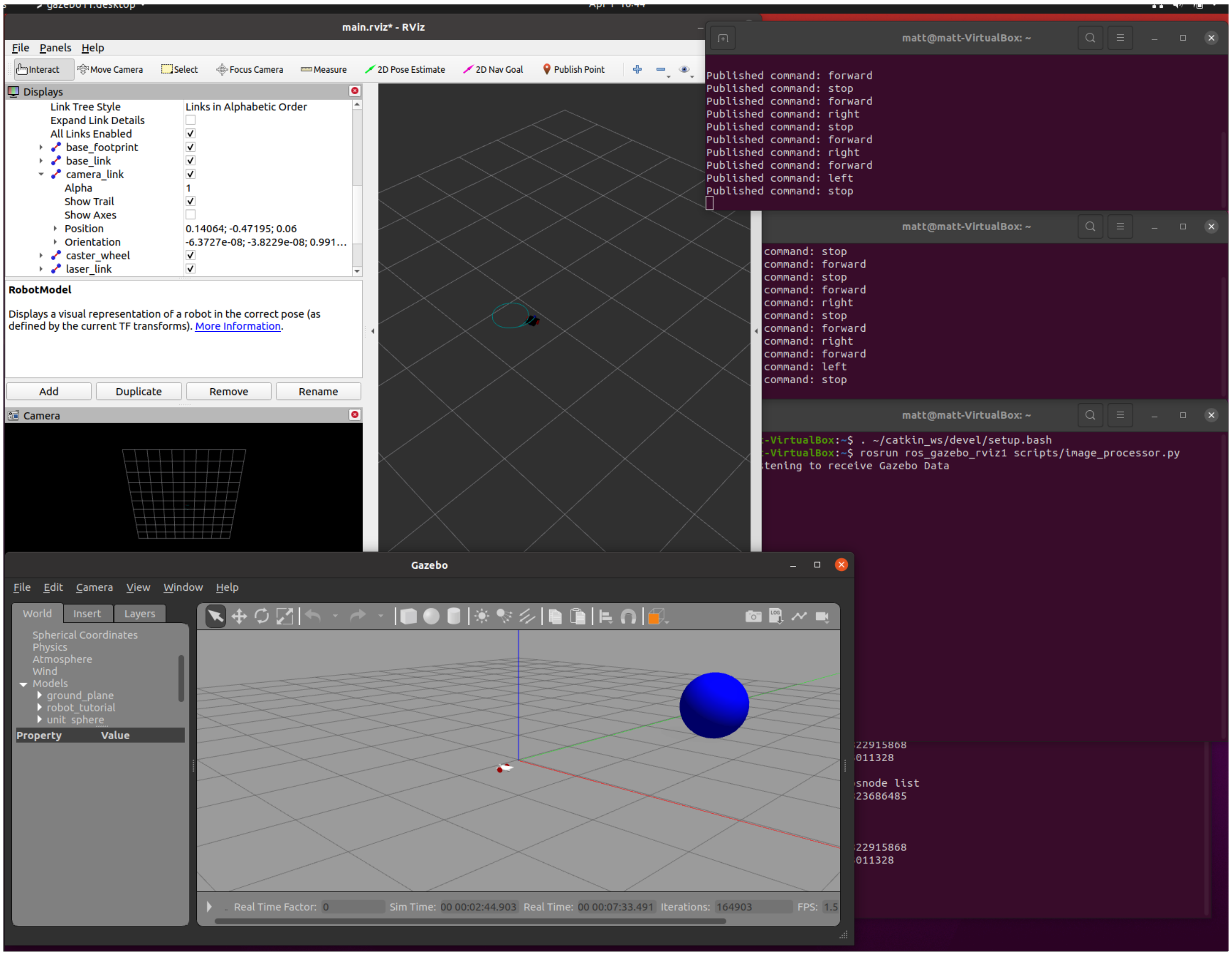
    iv. And the robot_state_publisher is sending back live robot feedback from Gazebo.

2. And check topics:

    a.
    ```
    matt@matt-VirtualBox:~$ rostopic list
    /camera_info
    /clicked_point
    /clock
    /cmd_vel
    /command
    /gazebo/link_states
    /gazebo/model_states
    /gazebo/parameter_descriptions
    /gazebo/parameter_updates
    /gazebo/performance_metrics
    /gazebo/set_link_state
    /gazebo/set_model_state
    /image_raw
    /image_raw/compressed
    /image_raw/compressed/parameter_descriptions
    /image_raw/compressed/parameter_updates
    /image_raw/compressedDepth
    /image_raw/compressedDepth/parameter_descriptions
    /image_raw/compressedDepth/parameter_updates
    /image_raw/theora
    /image_raw/theora/parameter_descriptions
    /image_raw/theora/parameter_updates
    /image_width
    /initialpose
    /joint_states
    /move_base_simple/goal
    /odom
    /robot/camera1/parameter_descriptions
    /robot/camera1/parameter_updates
    /rosout
    /rosout_agg
    /scan
    /target_coord
    /target_radius
    /tf
    /tf_static
    ```

    b. Excellent, we have all the image data, robot pose, image coordinates, gazebo information, and robot commands. It’s all now communicating.

Next will do a driving demo to ensure everything is talking correctly:
Good, the robot is accepting driving and turning commands as expected



7. Appendix 1: Source Code

1. Gazebo Launch File

```
<launch>

  <!--Robot Description from URDF-->
  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
ros_gazebo_rviz1)/urdf/robot_tutorial.xacro"/>

  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher"/>

  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher"/>

  <!--RViz-->
```

```
  <node name="rviz" pkg="rviz" type="rviz" required="true" args="-d $(find
ros_gazebo_rviz1)/rviz/main.rviz" />

  <!--Gazebo empty world launch file-->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="debug" value="false" />
    <arg name="gui" value="true" />
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="false"/>
    <arg name="headless" value="false"/>
    <arg name="verbose" value="true"/>
  </include>

  <!--Gazebo Simulator-->
  <node name="spawn_model" pkg="gazebo_ros" type="spawn_model" args="-urdf -param
robot_description -model robot_tutorial" output="screen"/>

</launch>
```

2. **Drive Node**

```
#!/usr/bin/env python

import rospy # Python library for ROS
from std_msgs.msg import String, UInt16 # String and Unsigned integer message types
from geometry_msgs.msg import Point, Twist # Point (x, y, z) message type
#import RPi.GPIO as GPIO # Raspberry i GPIO library

# Set the GPIO mode
#GPIO.setmode(GPIO.BCM)
#GPIO.setwarnings(False) # Disable GPIO warnings

# Set variables for the GPIO motor driver pins
motor_left_fw_pin  = 10
motor_left_bw_pin  = 9
motor_right_fw_pin = 8
motor_right_bw_pin = 7

# PWM signal frequency in Hz
pwm_freq = 2000 # Use between 2000 - 20000

# PWM % duty cycle (change these to the values that work best for you)
fw_bw_duty_cycle = 60
```

```
turn_duty_cycle  = 40

# Set the GPIO Pin mode to output
#GPIO.setup(motor_left_fw_pin, GPIO.OUT)
#GPIO.setup(motor_left_bw_pin, GPIO.OUT)
#GPIO.setup(motor_right_fw_pin, GPIO.OUT)
#GPIO.setup(motor_right_bw_pin, GPIO.OUT)

# Create PWM objects to handle GPIO pins with 'pwm_freq' frequency
#motor_left_fw  = GPIO.PWM(motor_left_fw_pin, pwm_freq)
#motor_left_bw  = GPIO.PWM(motor_left_bw_pin, pwm_freq)
#motor_right_fw = GPIO.PWM(motor_right_fw_pin, pwm_freq)
#motor_right_bw = GPIO.PWM(motor_right_bw_pin, pwm_freq)

# Start PWM with a duty cycle of 0 by default
#motor_left_fw.start(0)
#motor_left_bw.start(0)
#motor_right_fw.start(0)
#motor_right_bw.start(0)

# Global variables for storing received ROS messages
received_command = ''
last_received_command = ''
received_coord = Point(0, 0, 0)
target_radius = None
MIN_TGT_RADIUS_PERCENT = 0.05
image_width = 0
CENTER_WIDTH_PERCENT = 0.30

# Publish the cmd_vel values
motor_command = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

def listener():
   # Initialize this node with a the name 'motor_driver'
   rospy.init_node('motor_driver', anonymous=True)

   # Subscribe to the '/command' topic
   rospy.Subscriber('/command', String, commandCallback)

   # Subscribe to the '/target_coord' topic
   rospy.Subscriber('/target_coord', Point, targetCoordCallback)

   # Subscribe to the '/target_radius' topic
   rospy.Subscriber('/target_radius', UInt16, targetRadiusCallback)
```

```
    # Subscribe to the '/image_width' topic
    rospy.Subscriber('/image_width', UInt16, imageWidthCallback)



    # Put this node in an inifinite loop to execute when new messages arrive
    rospy.spin()

# '/command' topic message handler
def commandCallback(message):
    global received_command
    global last_received_command

    received_command = message.data

    if received_command == 'forward':
        forward()
    elif received_command == 'backward':
        backward()
    elif received_command == 'left':
        left()
    elif received_command == 'right':
        right()
    elif received_command == 'stop':
        stopMotors()
    elif received_command == 'auto':
        autonomous()
    else:
        print('Unknown command!')

    if received_command != last_received_command:
        print('Received command: ' + received_command)
        last_received_command = received_command

# Follow the target in autonomous mode:
def autonomous():
    global image_width
    global target_radius
    global MIN_TGT_RADIUS_PERCENT

    if target_radius >= image_width*MIN_TGT_RADIUS_PERCENT and target_radius <=
image_width/3:
        if abs(received_coord.x) <= image_width*CENTER_WIDTH_PERCENT:
```

```
            forward()
        elif received_coord.x > 0:
            right()
        elif received_coord.x < 0:
            left()
    else:
        stopMotors()
        print('stopMotors')

# '/image_width' topic message handler
def imageWidthCallback(message):
    global image_width

    image_width = message.data

# '/target_radius' topic message handler
def targetRadiusCallback(message):
    global target_radius

    target_radius = message.data

# '/target_coord' topic message handler
def targetCoordCallback(message):
    # global received_coord

    received_coord.x = message.x
    received_coord.y = message.y
    # print("received_coord = ", received_coord.x, received_coord.y)

# Turn both motors forwards
def forward():
    motor_left_fw = fw_bw_duty_cycle
    motor_left_bw = 0
    motor_right_fw = fw_bw_duty_cycle
    motor_right_bw = 0

    #configure cmd_vel output
    cmd_vel = Twist()
    cmd_vel.linear.x = 0.1
    cmd_vel.angular.z = 0.0
    motor_command.publish(cmd_vel)

# Turn both motors backwards
def backward():
```

```
    motor_left_fw = 0
    motor_left_bw = fw_bw_duty_cycle
    motor_right_fw = 0
    motor_right_bw = fw_bw_duty_cycle

    #configure cmd_vel output
    cmd_vel = Twist()
    cmd_vel.linear.x = -0.1
    cmd_vel.angular.z = 0.0
    motor_command.publish(cmd_vel)

# Turn left motor backward, right motor forward
def left():
    motor_left_fw = 0
    motor_left_bw = turn_duty_cycle
    motor_right_fw = turn_duty_cycle
    motor_right_bw = 0

    #configure cmd_vel output
    cmd_vel = Twist()
    cmd_vel.linear.x = 0.1
    cmd_vel.angular.z = 0.5
    motor_command.publish(cmd_vel)

# Turn right motor backward, left motor forward
def right():
    motor_left_fw = turn_duty_cycle
    motor_left_bw = 0
    motor_right_fw = 0
    motor_right_bw = turn_duty_cycle

    #configure cmd_vel output
    cmd_vel = Twist()
    cmd_vel.linear.x = 0.1
    cmd_vel.angular.z = -0.5
    motor_command.publish(cmd_vel)

# Turn all motors off
def stopMotors():
    motor_left_fw = 0
    motor_left_bw = 0
    motor_right_fw = 0
    motor_right_bw = 0
```

```
    #configure cmd_vel output
    cmd_vel = Twist()
    cmd_vel.linear.x = 0.0
    cmd_vel.angular.z = 0.0
    motor_command.publish(cmd_vel)

if __name__ == '__main__':
    print('Ready to receive commands!')
    listener()
    print('Node is shutting down, stopping motors')
    stopMotors()
```

3. Command Node

```
#!/usr/bin/env python
# Parts of this code are based on the 'teleop_twist_keyboard' node

import rospy
from std_msgs.msg import String
import sys, select, termios, tty

command = 'stop'
last_command = 'stop'

msg = """
Reading from the keyboard and publishing to /command!
---------------------------
Moving around:
        i
   j    k    l
        ,

CTRL-C to quit
"""

def talker():
    global command
    global last_command

    pub = rospy.Publisher('/command', String, queue_size=10)
    rospy.init_node('keyb_commander', anonymous=True)
    rate = rospy.Rate(10) # 10hz

    print(msg)
```

```
    # rospy.rosinfo(msg)
    while not rospy.is_shutdown():
        key_timeout = 0.6
        k = getKey(key_timeout)

        if k == "i":
            command = 'forward'
        elif k == ",":
            command = 'backward'
        elif k == "j":
            command = 'left'
        elif k == "l":
            command = 'right'
        elif k == "k":
            command = 'stop'
        elif k == "a":
            command = 'auto'
        elif k == '\x03': # To detect CTRL-C
            break

        if command != last_command:
            print("Published command: " + command)
            last_command = command
        pub.publish(command)
        rate.sleep()

def getKey(key_timeout):
    tty.setraw(sys.stdin.fileno())
    rlist, _, _ = select.select([sys.stdin], [], [], key_timeout)
    if rlist:
        key = sys.stdin.read(1)
    else:
        key = ''
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key

if __name__ == '__main__':
    settings = termios.tcgetattr(sys.stdin)
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

4. **Image_Processor.py**

```
#!/usr/bin/env python

# Import libraries
import rospy # Python library for ROS
from sensor_msgs.msg import CompressedImage, CameraInfo # CompressedImage message
type
from geometry_msgs.msg import Point # Point (x, y, z) message type
from std_msgs.msg import UInt16, String # Unsigned integer message type
import cv2 # OpenCV library
from cv_bridge import CvBridge # Converts between OpenCV and ROS images
import time
from PIL import Image
import numpy as np

# Global constants and variables
NUM_FILT_POINTS      = 5 # Number of filtering points for the Moving Average Filter
DESIRED_IMAGE_HEIGHT = 240 # A smaller image makes the detection less CPU intensive

# A dictionary of two empty buffers (arrays) for the Moving Average Filter
filt_buffer = {'width':[], 'height':[]}

# A dictionary of general parameters derived from the camera image size,
# which will be populated later with the 'get_image_params' function
params = {'image_height':None, 'image_width': None,'resized_height':None,'resized_width':
None,
   'x_ax_pos':None, 'y_ax_pos':None, 'scaling_factor':None}

def listener():
   # Set the node's name
   rospy.init_node('detection_node', anonymous=True)

   # Subscribe to the '/command' topic
   rospy.Subscriber('/image_raw', UInt16, RawImageCallback, queue_size=10)

   # Subscribe to the '/camera_info' topic
   rospy.Subscriber('/camera_info', CameraInfo, CameraInfoCallback, queue_size=10)

   # Put this node in an inifinite loop to execute when new messages arrive
   #rospy.spin()

# '/image_raw' topic message handler
def callbackImage(message):
   global image_raw
```

```
    image_raw = message.data
    print(image_raw)

# '/image_raw' topic message handler
def callbackImageComp(message):
    global image_comp
    global np_img

    #image_comp = message.data

    #image_comp = cv2.imread(img)
    #print(image_comp)
    np_arr = np.fromstring(message.data, np.uint8)
    #print(np_arr)
    np_img = np_arr
    image_comp = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
    image_comp = np_arr
    #print(image_comp)

# '/camera_info' topic message handler
def CameraInfoCallback(message):
    global camera_info
    # all the camera data (header, height, width, distortion_model, D, K, R, P, binning_x,
binning_y, roi)
    camera_info = message
    # pull useful bits
    image_height = camera_info.height
    image_width = camera_info.width


def detect_target():
  """ Main entry function for this node. """
  global image_raw
  global image_comp
  global np_img

  # Publishes the video frames from the detection process
  # detect_image_pub = rospy.Publisher('detect_image', Image, queue_size=10)
  detect_image_pub = rospy.Publisher('detect_image/compressed', CompressedImage,
queue_size=10)

  # Publishes the (x, y, 0) coordinates for the detected target
  target_coord_pub = rospy.Publisher('/target_coord', Point, queue_size=10)
```

```
 #print(target_coord_pub)
 # Publishes the detected target's computed enclosing radius
 target_radius_pub = rospy.Publisher('/target_radius', UInt16, queue_size=10)
 #print(target_radius_pub)
 # Publishes the image frame width after scaling used in the detection process
 image_width_pub = rospy.Publisher('/image_width', UInt16, queue_size=10)

 # Set the node's name
 rospy.init_node('detection_node', anonymous=True)

 # The node will run 30 times per second
 rate = rospy.Rate(0.2) # 30 Hz

 # Subscribe to the '/command' topic
 rospy.Subscriber('/image_raw', UInt16, callbackImage)

 image_comp = rospy.Subscriber('/image_raw/compressed', CompressedImage,
callbackImageComp)
 #print(image_comp.data)

 # Subscribe to the '/camera_info' topic
 rospy.Subscriber('camera_info', CameraInfo, CameraInfoCallback, queue_size=10)

 # Create a VideoCapture object
 #vid_cam = cv2.VideoCapture(0) # '0''is de index for the default webcam

 # Check if the camera opened correctly
 #if vid_cam.isOpened() is False:
 #  print('[ERROR] Couldnt open the camera.')
 #  return

 #print('-- Camera opened successfully')

 # define image
 #image = image_raw

 # Compute general parameters
 #get_image_params(vid_cam)
 #print("-- Original image width, height: ", {params['image_width']}, {params['image_height']})

 # To convert between OpenCV and ROS images
 bridge = CvBridge()

 # While ROS is still running.
```

```
  while not rospy.is_shutdown():
      start_time = time.time()

      # Set the node's name
      rospy.init_node('detection_node', anonymous=True)

      # Subscribe to the '/command' topic
      rospy.Subscriber('/image_raw/compressed', CompressedImage, callbackImageComp)

      pix = np.array(image_comp)
      print(pix)
      #img = cv2.resize(image_comp, (240,240))
      img = image_comp
      print("img: ", img)

      # Compute general parameters
      get_image_params(img)

      # Get the target coordinates
      tgt_cam_coord, frame, contour, radius = get_target_coordinates(img)

      # If a target was found, filter their coordinates
      if tgt_cam_coord['width'] is not None and tgt_cam_coord['height'] is not None:
          # Apply Moving Average filter to target camera coordinates
          tgt_filt_cam_coord = moving_average_filter(tgt_cam_coord)

      # No target was found, set target camera coordinates to the Cartesian origin,
      # so the drone doesn't move
      else:
          # The Cartesian origin is where the x and y Cartesian axes are located
          # in the image, in pixel units
          tgt_cam_coord = {'width':params['y_ax_pos'], 'height':params['x_ax_pos']} # Needed just
for drawing objects
          tgt_filt_cam_coord = {'width':params['y_ax_pos'], 'height':params['x_ax_pos']}

      # Convert from camera coordinates to Cartesian coordinates (in pixel units)
      tgt_cart_coord = {'x':(tgt_filt_cam_coord['width'] - params['y_ax_pos']),
                        'y':(params['x_ax_pos'] - tgt_filt_cam_coord['height'])}

      # Draw objects over the detection image frame just for visualization
      frame = draw_objects(tgt_cam_coord, tgt_filt_cam_coord, frame, contour)

      # Publish the detection image after convertin from OpenCV to ROS
      # detect_image_pub.publish(bridge.cv2_to_imgmsg(frame))
```

```
        detect_image_pub.publish(bridge.cv2_to_compressed_imgmsg(frame))

        # Publish the detected target's coordinates (x, y, 0)
        tgt_coord_msg = Point(tgt_cart_coord['x'], tgt_cart_coord['y'], 0)
        target_coord_pub.publish(tgt_coord_msg)
        print(tgt_coord_msg)
        # Publish de detected target's enclosing radius
        target_radius_pub.publish(radius)
        print(radius)
        # Publish the image frame's resized width
        image_width_pub.publish(params['resized_width'])

        # Show the detection image frame on screen
        # Optionally you can comment this line when running this node remotely through SSH:
        cv2.imshow("Detect and Track", frame)

        delta_time = end_time = time.time()
        detection_time = round(end_time-start_time, 3)
        print("Detection time: " + str(detection_time ))

        # Catch aborting key from computer keyboard
        key = cv2.waitKey(1) & 0xFF
        # If the 'q' key is pressed, break the 'while' infinite loop
        if key == ord("q"):
            break

        # Sleep just enough to maintain the desired rate
        rate.sleep()

def get_image_params(image):
  """ Computes useful general parameters derived from the camera image size."""

  # Grab a frame and get its size
  #is_grabbed, frame = vid_cam.read()
  frame=image
  try:
    params['image_height'], params['image_width'], _ = image.shape
  except:
    params['image_height'] = 240
    params['image_width'] = 240
  # Compute the scaling factor to scale the image to a desired size
  if params['image_height'] != DESIRED_IMAGE_HEIGHT:
      # Rounded scaling factor. Convert 'DESIRED_IMAGE_HEIGHT' to float or the division will
throw zero
```

```
    params['scaling_factor'] = round((float(DESIRED_IMAGE_HEIGHT) /
params['image_height']), 3)
  else:
    params['scaling_factor'] = 1

  print("params['scaling_factor']: ", params['scaling_factor'])
  print("params['scaling_factor']: ", DESIRED_IMAGE_HEIGHT / params['image_height'])
  # Compute resized width and height and resize the image
  params['resized_width'] = int(params['image_width'] * params['scaling_factor'])
  params['resized_height'] = int(params['image_height'] * params['scaling_factor'])
  dimension = (params['resized_width'], params['resized_height'])
  # dimension = (int(params['resized_width']), int(params['resized_height']))
  #frame = cv2.resize(frame, dimension, interpolation = cv2.INTER_AREA)

  # Compute the position for the X and Y Cartesian coordinates in camera pixel units
  params['x_ax_pos'] = int(params['resized_height']/2 - 1)
  params['y_ax_pos'] = int(params['resized_width']/2 - 1)

  return

def get_target_coordinates(image):
    """ Detects a target by using color range segmentation and returns its 'camera pixel'
coordinates."""

    # Use the 'threshold_inRange.py' script included with the code to get
    # your own bounds with any color
    # To detect a blue target:
    HSV_LOWER_BOUND = (107, 119, 41)
    HSV_UPPER_BOUND = (124, 255, 255)

    # Grab a frame in BGR (Blue, Green, Red) space color
    #is_grabbed, frame = vid_cam.read()
    frame = image

    # Resize the image frame for the detection process, if needed
    if params['scaling_factor'] != 1:
        dimension = (params['resized_width'], params['resized_height'])
        frame = cv2.resize(frame, dimension, interpolation = cv2.INTER_AREA)

    # Blur the image to remove high frequency content
    blurred = cv2.GaussianBlur(frame, (11, 11), 0)

    # Change color space from BGR to HSV
    hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
```

```
# Histogram equalisation to minimize the effect of variable lighting
# hsv[:, :, 0] = cv2.equalizeHist(hsv[:, :, 0]) # on the H-channel
# hsv[:, :, 1] = cv2.equalizeHist(hsv[:, :, 1]) # on the S-channel
# hsv[:, :, 2] = cv2.equalizeHist(hsv[:, :, 2]) # on the V-channel

# Get a mask with all the pixels inside our defined color boundaries
mask = cv2.inRange(hsv, HSV_LOWER_BOUND, HSV_UPPER_BOUND)

# Erode and dilate to remove small blobs
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)

# Find all contours in the masked image
_, contours, _ = cv2.findContours(mask,
      cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Centroid coordinates to be returned:
cX = None
cY = None

# To save the larges contour, presumably the detected object
largest_contour = None
tgt_radius = 0

# Check if at least one contour was found
if len(contours) > 0:
   # Get the largest contour of all posibly detected
   largest_contour = max(contours, key=cv2.contourArea)

   # Compute the radius of an enclosing circle aorund the largest contour
   (x,y), tgt_radius  = cv2.minEnclosingCircle(largest_contour)

   center = (int(x),int(y))
   tgt_radius = int(tgt_radius)
   # cv2.circle(frame, center, tgt_radius , (3, 186, 252), 3)

   # Compute contour raw moments
   M = cv2.moments(largest_contour)
   # Get the contour's centroid
   cX = int(M["m10"] / M["m00"])
   cY = int(M["m01"] / M["m00"])

# Return centroid coordinates (camera pixel units), the analized frame and the largest contour
```

```
    return {'width':cX, 'height':cY}, frame, largest_contour, tgt_radius


def moving_average_filter(coord):
    """ Applies Low-Pass Moving Average Filter to a pair of coordinates."""

    # Append new coordinates to filter buffers
    filt_buffer['width'].append(coord['width'])
    filt_buffer['height'].append(coord['height'])

    # If the filters were full already with a number of NUM_FILT_POINTS values,
    # discard the oldest value (FIFO buffer)
    if len(filt_buffer['width']) > NUM_FILT_POINTS:
        filt_buffer['width'] = filt_buffer['width'][1:]
        filt_buffer['height'] = filt_buffer['height'][1:]

    # Compute filtered camera coordinates
    N = len(filt_buffer['width']) # Get the number of values in buffers (will be <
NUM_FILT_POINTS at the start)

    # Sum all values for each coordinate
    w_sum = sum( filt_buffer['width'] )
    h_sum = sum( filt_buffer['height'] )
    # Compute the average
    w_filt = int(round(w_sum / N))
    h_filt = int(round(h_sum / N))

    # Return filtered coordinates as a dictionary
    return {'width':w_filt, 'height':h_filt}


def draw_objects(cam_coord, filt_cam_coord, frame, contour):
  """ Draws visualization objects from the detection process.
  Position coordinates of every object are always in 'camera pixel' units"""

  # Draw the Cartesian axes
  cv2.line(frame, (0, params['x_ax_pos']), (params['resized_width'], params['x_ax_pos']), (0, 128,
255), 1)
  cv2.line(frame, (params['y_ax_pos'], 0), (params['y_ax_pos'], params['resized_height']), (0,
128, 255), 1)
  cv2.circle(frame, (params['y_ax_pos'], params['x_ax_pos']), 1, (255, 255, 255), -1)

  # Draw the detected object's contour, if any
  if contour is not None:
```

```
    cv2.drawContours(frame, [contour], -1, (0, 255, 0), 2)

  # Compute Cartesian coordinates of unfiltered detected object's centroid
  x_cart_coord = cam_coord['width'] - params['y_ax_pos']
  y_cart_coord = params['x_ax_pos'] - cam_coord['height']

  # Compute Cartesian coordinates of filtered detected object's centroid
  x_filt_cart_coord = filt_cam_coord['width'] - params['y_ax_pos']
  y_filt_cart_coord = params['x_ax_pos'] - filt_cam_coord['height']

  # Draw unfiltered centroid as a red dot, including coordinate values
  cv2.circle(frame, (cam_coord['width'], cam_coord['height']), 5, (0, 0, 255), -1)
  cv2.putText(frame, str(x_cart_coord) + ", " + str(y_cart_coord),
    (cam_coord['width'] + 25, cam_coord['height'] - 25), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(0, 0, 255), 1)

  # Draw filtered centroid as a yellow dot, including coordinate values
  cv2.circle(frame, (filt_cam_coord['width'], filt_cam_coord['height']), 5, (3, 186, 252), -1)
  cv2.putText(frame, str(x_filt_cart_coord) + ", " + str(y_filt_cart_coord),
    (filt_cam_coord['width'] + 25, filt_cam_coord['height'] - 25),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (3, 186, 252), 1)

  return frame # Return the image frame with all drawn objects


if __name__ == '__main__':
 # try:
  print("Start Listening to receive Gazebo Data")
  #listener()
  print("Prepare for Autonomous Target Detection")
  detect_target()
  #except rospy.ROSInterruptException:
  #  pass
```

5. Robot_tutorial.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="robot_tutorial">

  <xacro:property name="base_width" value="0.16"/>
  <xacro:property name="base_len" value="0.16"/>
  <xacro:property name="wheel_radius" value="0.035"/>
  <xacro:property name="base_wheel_gap" value="0.007"/>
  <xacro:property name="wheel_separation" value="0.15"/>
```

```
  <xacro:property name="wheel_joint_offset" value="0.02"/>

  <xacro:property name="caster_wheel_radius" value="${wheel_radius/2}"/>
  <xacro:property name="caster_wheel_mass" value="0.001"/>
  <xacro:property name="caster_wheel_joint_offset" value="-0.052"/>

  <!--Color Properties-->
  <material name="blue">
    <color rgba="0 0 0.8 1"/>
  </material>
  <material name="black">
    <color rgba="0 0 0 1"/>
  </material>
  <material name="white">
    <color rgba="1 1 1 1"/>
  </material>
  <material name="red">
    <color rgba="0.8 0.0 0.0 1.0"/>
  </material>

  <!--Interial macros-->
  <xacro:macro name="cylinder_inertia" params="m r h">
    <inertial>
      <mass value="${m}"/>
      <inertia ixx="${m*(3*r*r+h*h)/12}" ixy = "0" ixz = "0" iyy="${m*(3*r*r+h*h)/12}" iyz = "0"
izz="${m*r*r/2}"/>
    </inertial>
  </xacro:macro>

  <xacro:macro name="box_inertia" params="m w h d">
    <inertial>
      <mass value="${m}"/>
      <inertia ixx="${m / 12.0 * (d*d + h*h)}" ixy="0.0" ixz="0.0" iyy="${m / 12.0 * (w*w + h*h)}"
iyz="0.0" izz="${m / 12.0 * (w*w + d*d)}"/>
    </inertial>
  </xacro:macro>

  <xacro:macro name="sphere_inertia" params="m r">
    <inertial>
      <mass value="${m}"/>
      <inertia ixx="${2.0*m*(r*r)/5.0}" ixy="0.0" ixz="0.0" iyy="${2.0*m*(r*r)/5.0}" iyz="0.0"
izz="${2.0*m*(r*r)/5.0}"/>
    </inertial>
  </xacro:macro>
```

```
<!--Base Footprint-->
<link name="base_footprint">
  <xacro:box_inertia m="10" w="0.001" h="0.001" d="0.001"/>
  <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
          <box size="0.001 0.001 0.001" />
      </geometry>
  </visual>
</link>

<!--Base link-->
<link name="base_link">
  <xacro:box_inertia m="10" w="${base_len}" h="${base_width}" d="0.01"/>
  <visual>
    <geometry>
      <box size="${base_len} ${base_width} 0.01"/>
    </geometry>
    <material name="black"/>
  </visual>
  <collision>
    <geometry>
      <box size="${base_len} ${base_width} 0.01"/>
    </geometry>
  </collision>
</link>

<!--base_link to base_footprint Joint-->
<joint name="base_link_joint" type="fixed">
  <origin xyz="0 0 ${wheel_radius + 0.005}" rpy="0 0 0" />
  <parent link="base_footprint"/>
  <child link="base_link" />
</joint>

<!--Wheel link & joint macro-->
<xacro:macro name="wheel" params="prefix reflect">
  <link name="${prefix}_wheel">
    <visual>
      <origin xyz="0 0 0" rpy="${pi/2} 0 0"/>
      <geometry>
        <cylinder radius="${wheel_radius}" length="0.005"/>
      </geometry>
      <material name="red"/>
```

```
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="${pi/2} 0 0"/>
        <geometry>
          <cylinder radius="${wheel_radius}" length="0.005"/>
        </geometry>
      </collision>
      <xacro:cylinder_inertia m="10" r="${wheel_radius}" h="0.005"/>
    </link>

    <joint name="${prefix}_wheel_joint" type="continuous">
      <axis xyz="0 1 0" rpy="0 0 0" />
      <parent link="base_link"/>
      <child link="${prefix}_wheel"/>
      <origin xyz="${wheel_joint_offset} ${((base_width/2)+base_wheel_gap)*reflect} -0.005"
rpy="0 0 0"/>
    </joint>
  </xacro:macro>

  <!--Create Left & Right Wheel links/joints-->
  <xacro:wheel prefix="left" reflect="1"/>
  <xacro:wheel prefix="right" reflect="-1"/>

  <!--Caster Wheel Link-->
  <link name="caster_wheel">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <sphere radius="${caster_wheel_radius}"/>
      </geometry>
      <material name="blue"/>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <sphere radius="${caster_wheel_radius}"/>
      </geometry>
    </collision>
    <xacro:sphere_inertia m="10" r="${caster_wheel_radius}"/>
  </link>

  <!--Caster Wheel Joint-->
  <joint name="caster_wheel_joint" type="continuous">
    <axis xyz="0 1 0" rpy="0 0 0" />
```

```
  <parent link="base_link"/>
  <child link="caster_wheel"/>
  <origin xyz="${caster_wheel_joint_offset} 0 -${caster_wheel_radius+0.005}" rpy="0 0 0"/>
</joint>

<!-- Laser Link-->
<link name="laser_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.025 0.025 0.025" />
    </geometry>
    <material name="blue"/>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.025 0.025 0.025" />
    </geometry>
  </collision>
  <xacro:box_inertia m="1" w="0.1" h="0.1" d="0.1" />
</link>

<!--Laser Joint-->
<joint name="laser_joint" type="fixed">
  <axis xyz="0 1 0" />
  <origin xyz="0.075 0 0.02" rpy="0 0 0" />
  <parent link="base_link"/>
  <child link="laser_link"/>
</joint>

<!-- Camera Link-->
<link name="camera_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.025 0.025 0.025" />
    </geometry>
    <material name="red"/>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.025 0.025 0.025" />
```

```
    </geometry>
  </collision>
  <xacro:box_inertia m="1" w="0.1" h="0.1" d="0.1" />
</link>

<!--Camera Joint-->
<joint name="camera_joint" type="fixed">
  <axis xyz="0 1 0" />
  <origin xyz="0.075 0 0.02" rpy="0 0 0" />
  <parent link="base_link"/>
  <child link="camera_link"/>
</joint>

<xacro:include filename="$(find ros_gazebo_rviz1)/urdf/robot_tutorial_gazebo.xacro"/>

</robot>
```

6. **Robot_tutorial_gazebo.xacro**

```
<?xml version="1.0"?>
<robot>

  <gazebo>
    <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
      <alwaysOn>false</alwaysOn>
      <legacyMode>false</legacyMode>
      <updateRate>20</updateRate>
      <leftJoint>left_wheel_joint</leftJoint>
      <rightJoint>right_wheel_joint</rightJoint>
      <wheelSeparation>${wheel_separation}</wheelSeparation>
      <wheelDiameter>${wheel_radius * 2}</wheelDiameter>
      <torque>20</torque>
      <commandTopic>/cmd_vel</commandTopic>
      <odometryTopic>/odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <robotBaseFrame>base_footprint</robotBaseFrame>
    </plugin>
  </gazebo>

  <gazebo reference="base_link">
    <material>Gazebo/White</material>
  </gazebo>
  <gazebo reference="left_wheel">
```

```
  <material>Gazebo/Red</material>
</gazebo>
<gazebo reference="right_wheel">
  <material>Gazebo/Red</material>
</gazebo>

<gazebo reference="laser_link">
  <sensor type="ray" name="laser_sensor">
    <pose>0 0 0 0 0 0 </pose>
    <visualize>false</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>5</samples>
          <min_angle>-0.0349066</min_angle>
          <max_angle>0.0349066</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.10</min>
        <max>30.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
      <topicName>/scan</topicName>
      <frameName>laser_link</frameName>
    </plugin>
  </sensor>
</gazebo>

<gazebo reference="camera_link">
  <sensor type="camera" name="camera1">
    <update_rate>1.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>240</width>
```

```
        <height>240</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
      <noise>
        <type>gaussian</type>
        <!-- Noise is sampled independently per pixel on each frame.
             That pixel's noise value is added to each of its color
             channels, which at that point lie in the range [0,1]. -->
        <mean>0.0</mean>
        <stddev>0.007</stddev>
      </noise>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>robot/camera1</cameraName>
      <imageTopicName>/image_raw</imageTopicName>
      <cameraInfoTopicName>/camera_info</cameraInfoTopicName>
      <frameName>camera_link</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>


 </gazebo>

</robot>
```